Some Prospects for Efficient Fixed Parameter Algorithms*

Rolf Niedermeier†
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
niedermr@informatik.uni-tuebingen.de

Abstract

Recent time has seen quite some progress in the development of exponential time algorithms for *NP*-hard problems, where the base of the exponential term is fairly small. These developments are also tightly related to the theory of fixed parameter tractability. In this incomplete survey, we explain some basic techniques in the design of *efficient* fixed parameter algorithms, discuss deficiencies of parameterized complexity theory, and try to point out some future research challenges. The focus of this paper is on the design of efficient algorithms and not on a structural theory of parameterized complexity. Moreover, our emphasis will be laid on two exemplifying issues: Vertex Cover and MaxSat problems.

*A shorter version of this paper appears as an invited talk in the proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'98), Springer, *LNCS*, held in Jasna, Slovakia, November 21–27, 1998.

†Supported by a Feodor Lynen fellowship of the Alexander von Humboldt-Stiftung, Bonn, and the Center for Discrete Mathematics, Theoretical Computer Science and Applications (DIMATIA), Prague. Author's address in 1998: DIMATIA MFF UK, Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic.

1 Introduction

How to cope with intractability? This is one of the most important problems in the theory and practice of computer science. Several methods to deal with this problem have been developed: approximation, average case analysis, randomization, and heuristics. All of them have their drawbacks as there are hardness of approximability, lack of mathematical tools and results, limited power of the method itself, or the lack of provable performance guarantees at all. Parameterization, whose cantus firmus can be characterized by the words "not all forms of intractability are created equal" [24], is another proposal how to cope with intractability in some cases. This is the basic subject of this paper.

Many hard computational problems have the following general form: given an object x and a natural number k, does x have some property that depends on k? For instance, the *NP*-complete Vertex Cover problem is: given an undirected graph $G = (V, E)$ and a natural number k, does G have a vertex cover of size at most k? Herein, a vertex cover is a subset of vertices $C \subseteq V$ such that each edge in E has at least one of its endpoints in C. In parameterized complexity theory, this natural number k is called the *parameter*. In many applications, the parameter k can be considered to be "very small" in comparison with the size $|x|$ of the given object x. Hence, it may be of high interest to ask whether that problems, which usually are *NP*-hard, have deterministic algorithms that *only* are exponential with respect to k, but polynomial with respect to $|x|$.

Parameterized complexity, as mainly developed by Downey and Fellows [1, 20, 21, 22, 23, 24, 25], is the perhaps latest approach to attack problems that are (worst case) intractable. The basic observation is that for many hard problems the seemingly inherent "combinatorial explosion" can be restrained to a "small part" of the input, the parameter. So, for instance, the *NP*-complete Vertex Cover problem allows for an algorithm with running time $O(kn + (1.3248)^k k^2)$ [5], where the parameter k is a bound on the maximum size of the vertex cover set we are looking for and n is the number of vertices of the given graph. The fundamental assumption is $k \ll n$. As can easily be seen, this yields an efficient, practical algorithm if only small values of k are involved. In this paper, we focus on issues concerning the development of *efficient* fixed parameter algorithms. However, there are also tight relations to the somewhat more general problem of designing exponential time algorithms with "small" exponential terms.

The writing of this paper was stimulated by the following conception: It is widely agreed that the notion of P versus NP reasonably reflects the difference between tractable and intractable problems. Why? Does an algorithm with running time n^{100}, putting the corresponding problem into P, have practical use? In general, no. The general observation, however, is that most problems in P in fact have $O(n^3)$ algorithms or better [31], which is not that enormous. Of course, from a practical point of view, this may still be unacceptable and usually the ultimate goal are linear or quasilinear time algorithms with small constant factors. For parameterized complexity, expressed conservatively, such an observation is hard to make. Problems are called fixed parameter tractable if they have running time $f(k)n^{O(1)}$ for an arbitrary function f only depending on k. Unfortunately, this $f(k)$ usually cannot be bounded so nicely as in the case of Vertex Cover (where $f(k) = (1.3248)^k$ [5][1], which, very recently, was further improved to $(1.2918)^k$ [50]), but grows much faster (e.g., still giving a harmless example, $f(k) = 11^k$ for the Planar Dominating Set problem [23]), making the fixed parameter tractable algorithm already impractical for small values of k. This might be one of the, so far, main deficiencies of parameterized complexity theory. Here, we will survey and explore some results directed to "efficient" fixed parameter tractability as represented by Vertex Cover. In particular, our main focus is on two elementary techniques used in the design of efficient fixed parameter algorithms: kernelization and bounded search trees.

We assume the reader to be familiar with basic notions from algorithms and complexity as, e.g., provided by the text books [12, 16, 31, 46, 51]. We omit material on graph minors, bounded treewidth algorithmics etc., which, on the one hand, play an important role in fixed parameter tractability theory, but, on the other hand, play a minor (sic!) role for the restricted point of view we are taking here—*elementary* methods in designing *efficient* fixed parameter algorithms. Let us mention in passing, however, that for graph problems graph minor theory is one of the main tools for showing fixed parameter tractability [24, 55]: If a graph

[1] Note that, according to the above, we actually have running time $O(kn + (1.3248)^k k^2)$. Assuming, however, $k \ll n$ it is easy to see that this means also a bound of the form $O(f(k)n^{O(1)})$.

class is minor closed, then this implies fixed parameter tractability of the corresponding problem. For instance, consider the class of graphs having a vertex cover of size at most k, which is closed under taking minors. Consequently, graph minor theory tells us that the problem is fixed parameter tractable. In addition, in the context of bounded treewidth there has been proposed a "design methodology that for many NP-hard problems results in algorithms with time complexity linear in the size of the input graph and only exponential in its treewidth, lowering the exponent of previously known solutions" [58]. Finally, let us mention the existence of a further general FPT method that uses hashing and is called "color-coding," developed by Alon *et al.* [2].

The paper is structured as follows. In the next section, we very briefly provide a general overview on some main topics and ideas of parameterized complexity theory. In Section 3, we take a closer look at the concept of "fixed parameter tractability" and its criticism, thus providing the basic motivation for this paper. Turning to the main approach of theoretical computer science in dealing with intractability, that is, approximation, in Section 4, we sketch some known relations between approximation algorithms and parameterized complexity. In Section 5, based on the Vertex Cover problem, we explain the two basic techniques, kernelization and search trees. We present the basic ideas behind the best known Vertex Cover algorithms and also discuss related approaches in solving important problems from reconfigurable VLSI. In Section 6, we discuss efficient fixed parameter algorithms for the maximum satisfiability problem. We end the paper by drawing some general conclusions.

2 A crash course in parameterized complexity

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E and a natural number k, the NP-complete *Vertex Cover* problem is to determine whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices (where k is a given natural number) such that each edge in E has at least one of its endpoints in C. Vertex Cover is *fixed parameter tractable*: There is an algorithm solving it in time

$O(kn + (1.3248)^k k^2)$ [5], making it efficiently solvable for reasonably small values of k. By way of contrast, consider the also NP-complete *Clique* problem: Given an undirected graph $G = (V, E)$, Clique asks whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices (where k is a given natural number) such that C forms a clique by having all possible edges between the vertices in C. Clique appears to be *fixed parameter intractable*: It is *not* known whether it can be solved in time $f(k)n^{O(1)}$, where f might be an arbitrarily fast growing function only depending on k [21]. Moreover, unless $P = NP$, the well-founded conjecture is that no such algorithm exists. Therefore, the best known algorithm solving Clique runs in time $O(n^{ck/3})$ [48], where c is the exponent on the time bound for multiplying two integer $n \times n$ matrices (currently best known, $c = 2.376\ldots$, see [15]). Note that n^k is trivial. The decisive point is that k appears in the exponent of n, and there seems to be no way "to shift the combinatorial explosion only into k," independent from n [24].

The observation that NP-complete problems like Vertex Cover and Clique behave completely differently in a "parameterized sense" lies at the very heart of parameterized complexity, which was pioneered by Downey and Fellows and some of their co-authors [21, 23, 24, 25, 54]. In this paper, we will focus on the world of fixed parameter tractable problems as, e.g., exhibited by Vertex Cover. Hence, here we only briefly sketch some very basics from the theory of parameterized intractability in order to provide some background on parameterized complexity theory and the ideas behind. For any further details and more discussion, we refer to the extensive literature, e.g., [23, 24, 25, 54].

Attempts to prove nontrivial, absolute lower bounds on the computational complexity of problems have made relatively little progress [10]. Hence, it is not surprising that up to now there is no proof that no $f(k)n^{O(1)}$ time algorithm for Clique exists. In a more complexity-theoretic language, where the class of parameterized problems that can be solved in deterministic time $f(k)n^{O(1)}$ is called FPT, this can be rephrased by saying that it is unknown whether Clique $\in FPT$. The complexity class FPT is called the set of *fixed parameter tractable* problems. Analogously to classical complexity theory, Downey and Fellows developed some way out of this quandary by providing a completeness program. However, the completeness theory of parameterized intractability involves significantly more technical effort. We briefly sketch some integral parts of this theory in the following.

To start with a completeness theory, we first need a reducibility concept: Let $L, L' \subseteq \Sigma^* \times \mathbf{N}$ be two parameterized languages. For example, in the case of Clique the first component is the input graph coded over some alphabet Σ and the second component is the natural number k, that is, the parameter. For complexity theory people, we mention in passing that the parameter k usually is encoded in unary as part of the input. We say that L *reduces to* L' *by a standard parameterized* m*-reduction* if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbf{N}$ to $\mathbf{N}$ and a function $(x, k) \mapsto x'$ from $\Sigma^* \times \mathbf{N}$ to Σ^* such that

1. $(x, k) \mapsto x'$ is computable in time $k''|x|^c$ for some constant c and
2. $(x, k) \in L$ iff $(x', k') \in L'$.

Notably, most reductions from classical complexity turn out *not* to be parameterized [24]. For instance, the well-known reduction from Independent Set to Vertex Cover (see [51]) is not a parameterized one. On the other hand, the reduction from Independent Set to Clique actually turns out to be also a parameterized one.

Now, the "lowest class of parameterized intractability", so-called $W[1]$, can be defined as the class of languages that reduce by a standard parameterized m-reduction to Clique. Hence, Clique is $W[1]$*-complete*. Independent Set is also $W[1]$-complete. A further, interesting $W[1]$-complete problem is *Weighted q-CNF-Sat*: Given a boolean formula F in conjunctive normal form and a positive integer k, does F have a truth assignment of weight k? Herein, the weight of a truth assignment simply is the number t of variables set true. Downey and Fellows provide an extensive list of many more $W[1]$-complete problems [21, 24].

As a matter of fact, a whole hierarchy of parameterized intractability can be defined, $W[1]$ only being the lowest level. In general, the classes $W[t]$ are defined based on "logical depth" (i.e., the number of alternations between unbounded fan-in And- and Or-gates) in boolean circuits. We omit any further details in this direction and just refer to the new monograph [24] or the many papers published on this topic, e.g., [1, 20, 21, 22, 23]. There exists a very rich structural theory of parameterized complexity, somewhat similar to classical complexity. Observe, however, that in some respects parameterized complexity appears to be in a sense "orthogonal" to classical complexity: For instance, the so-called problem of computing the V-C dimension from learning theory [8, 53], which is not known (and not believed) to be NP-hard, is $W[1]$-complete [19, 23]. Thus, although in the classical sense it appears to be easier than Vertex Cover (which is NP-complete), it appears to be exactly vice versa in the parameterized sense, because Vertex Cover is in FPT.

From a practical point of view, it is probably sufficient to distinguish between $W[1]$-hardness and membership in FPT. So, for an algorithm designer not being able to show fixed-parameter tractability of a problem, it may be sufficient to give a reduction from Clique or Weighted q-CNF-Sat to the given problem, using a standard parameterized m-reduction. This then gives a concrete indication that, unless $P = NP$, the problem is unlikely to allow for an $f(k)n^{O(1)}$ time algorithm. One circumstantial evidence for this is the result showing that the equality of $W[1]$ and FPT would imply a time $2^{o(n)}$ algorithm for the NP-complete 3-CNF-Sat problem [1, 24], which would mean a breakthrough in computational complexity theory.

In the remainder of this paper, however, we concentrate on the world inside FPT and the potential it carries for improvements and future research. Lots of problems termed fixed parameter tractable by the theory still wait for a proof of real "parameterized efficiency." In addition, there seem to be plenty of fields like computational biology or VLSI design, offering natural parameterized problems with efficient fixed parameter algorithms still to be discovered (also see [24, 25, 54]).

3 Interpreting fixed parameter tractability

There is a Vertex Cover algorithm running in time $O(kn + f(k))$, where $f(k) = O((1.2918)^k k^2)$ [50]. So, even for values like $k = 70$, this still makes an efficient algorithm, giving this result potential for practical importance. On the other hand, in the definition of FPT, $f(k)$ may take unreasonably large values, e.g.,

$$2^{2^{2^{2^{2^{2^{2^{k}}}}}}}.$$

Even a less enormous value like $f(k) = 11^k$ for the Planar Dominating Set problem [23] only provides efficient algorithms for quite small values of k. Downey and Fellows [24] introduced so-called klam values to address this. The *klam* value of an algorithm A solving a parameterized problem L is defined to be the largest k such that

1. L can be solved by A in time $f(k) + n^{O(1)}$ and
2. $f(k) \leq U$, where U is some reasonable absolute bound on the maximum number of steps of any computation, e.g., $U = 10^{20}$.

For example, using $U = 10^{20}$, the current klam value for Vertex Cover is approximately 140. Unfortunately, for few parameterized problems klam values of comparable high quality are known. Hence, an important algorithmic challenge concerning FPT problems is to provide klam values as large as possible.

To further substantiate the discussion before, let us briefly address another parameterized problem, namely maximum satisfiability for a boolean formula F in conjunctive normal form (CNF) with a constant number c of literals per clause (also see Section 6 for a more complete treatment). Here, for some time the best known algorithm had running time $O(2^{ck}m)$, where m is the number of clauses [14]. This, assuming a constant number of literals per clause, was first improved to $O(m + k\phi^k) \approx O(m + k(1.6181)^k)$, where ϕ is the golden ratio [45][2], and very recently was further improved to $O(m+k(1.3995)^k)$ [49]. Note that for the improvements it is not even necessary to assume a constant number of literals per clause. Then, however, the term m in the time bound has to be replaced by the formula length $|F|$ and the multiplicative factor k has to be replaced by k^2. Let us compare the exponential expressions involved in these three time bounds, setting $c = 2$. Table 3 provides these bounds for some reasonable values of k, implying that the klam value increases significantly and emphasizing the importance of the struggle to make the base of the exponential term as small as possible. So, the klam value for MaxSat corresponding to the three exponential algorithms referred to in Table 3, improves from approximately at most 35 to 90 to 125.

[2]Independently, this result was also achieved by Dantsin *et al.* [18] in the context of approximation algorithms.

Table 1: Comparing the efficiency of various MaxSat algorithms with respect to the exponential terms involved.

k	2^{2k}	$(1.6181)^k$	$(1.3995)^k$
10	$\approx 10^6$	≈ 124	29
20	$\approx 10^{12}$	≈ 15140	831
30	$\approx 10^{18}$	$\approx 1.9 \cdot 10^6$	≈ 24000
40	$\approx 10^{24}$	$\approx 2.3 \cdot 10^8$	$\approx 6.9 \cdot 10^5$
50	$\approx 10^{30}$	$\approx 2.9 \cdot 10^{10}$	$\approx 2.0 \cdot 10^7$
60	$\approx 10^{36}$	$\approx 3.5 \cdot 10^{12}$	$\approx 5.8 \cdot 10^8$

Finally, to also demonstrate the problematic nature of the comparison "fixed parameter tractable" versus "fixed parameter intractable", let us compare the functions 2^{2^k} and $n^k = 2^{(k \log n)}$. The first refers to fixed parameter tractability, the second to intractability. It is easy to verify that assuming input sizes n in the range from 10^3 up to 10^{15}, the value of k where 2^{2k} starts to exceed n^k is in the small range $\{ 6, 7, 8, 9 \}$. Hence, this shows how careful one has to be with the term fixed parameter tractable, since, in practice with reasonable input sizes, a fixed parameter intractable problem can easily turn out to have a still more efficient solution than a fixed parameter tractable one. In this context, note that the so far best known "fixed parameter intractable algorithm" for the Clique problem, running in time $O(n^{ck/3})$ [48], where c is the exponent on the time bound for multiplying two integer $n \times n$ matrices ($c = 2.376\ldots$, see [15]), still may be valuable. A perhaps even more striking example in this direction is that of computing treewidth. For constant k, there is a famous result giving a linear time algorithm to compute whether a graph has treewidth at most k [9]. However, this algorithm suffers from enormous constant factors (unless $k \leq 3$) and so the $O(n^{k+1})$ algorithm [3] is more practical.

4 Approximation and parameterization

In this section, we briefly discuss some relations of fixed parameter (in)tractability to one of the main topics of algorithms and complexity, that is, approximation. Polynomial time approximation algorithms and schemes are one of the major methods to cope with intractability [31]. Two recent surveys are available [17, 37]. Recently, deep results based on probabilistically checkable proofs have shown that many NP-hard optimization problems are also hard to approximate [4]. This gives rise to the general question of the nature of the relationship between fixed parameter tractability and approximability of problems. In this section, we will sketch some of the known results concerning this relationship and discuss implied consequences.

Results on the relationship between parameterized complexity and approximation come up in at least two ways—a more structural one and a more algorithmic one. We briefly study both of them, but afterwards direct our attention to the more algorithmic nature. Since this short section is anything but complete, for a more comprehensive treatment we refer to the literature [14, 24].

Optimization problems come in two forms: maximization and minimization problems. We concentrate on maximization problems, the minimization case works in analogy. A *maximization problem* is a 3-tuple (I, S, g), where I is the set of input instances, $S(x)$ is the set of feasible solutions for input $x \in I$, and $g(x, y) \in \mathbf{N}$ is the value for each $x \in I$ and $y \in S(x)$. The goal is to maximize $g(x, y)$. The *parameterized version* of a maximization problem is simply as follows. Given $x \in I$ and a positive integer k, is there a $y \in S(x)$ such that $g(x, y) \geq k$.

A maximization problem is *polynomial time approximable to a ratio r* if there is a polynomial time algorithm such that for all input instances $x \in I$ it produces a $y \in S(x)$ such that for the relative error it holds

$$\frac{\max(x)}{g(x, y)} \leq 1 + r,$$

where $\max(x)$ denotes the maximum, i.e., optimal, value for input x. A maximization problem has a *polynomial time approximation scheme (PTAS)* if for all $\epsilon > 0$ there is a polynomial time algorithm that produces a ratio ϵ approximation. Furthermore, it has a *fully polynomial* *time approximation scheme (FPTAS)* if it has a PTAS where, additionally, the running time of the algorithm is polynomial in the input size *as well as* in $1/\epsilon$.

It is not very difficult to prove the following interesting result [14]: If an NP optimization problem has an FPTAS, then it is in FPT. The basic idea of proof is to make use of the fact that if $\max(x)/g(x,y) \leq 1 + 1/(2k)$, this implies $\max(x) > k$ *iff* $g(x,y) > k$. Thus, we immediately get a number of knapsack-like and scheduling problems [38, 56] into FPT. The contrapositive consequences of this result appear to be still more important. As a corollary we get that the NP optimization problems that are $W[1]$-hard under the standard parameterized m-reduction have no FPTAS unless $W[1] = FPT$ [14]. Thus, the structural theory concerning the W-hierarchy surprisingly may give evidence on the non-approximability of optimization problems. In other words, proving $W[1]$-hardness can be seen as one way to show non-approximability. Further results of Cai and Chen show that the parameterized versions of all maximization problems in the class *MaxSNP*, introduced by Papadimitriou and Yannakakis [52], and all minimization problems in the class $MinF^{+}\Pi_1$ introduced by Kolaitis and Thakur [41] are in FPT. Hence, besides the above-mentioned, more structural issues, the subsequent questions arise:

1. Which problems in *MaxSNP* and $MinF^{+}\Pi_1$ admit *efficient* fixed parameter algorithms? What are the best time bounds?

2. More generally, can ideas from approximation algorithms be used for the design of efficient fixed parameter algorithms and vice versa?

3. For optimization problems a compendium of approximability results exists [17]. Will the future see something analogous for *efficient* fixed parameter tractability, giving the best achieved exponential time algorithms?

5 Vertex cover problems

The minimization problem Vertex Cover is surely one of the best explored parameterized problems. The problem instance is an undirected graph $G = (V, E)$ and a positive integer k, the question is whether there exists a "vertex cover set" $C \subseteq V$ with $|C| \leq k$ such that for all edges (u, v) in E, it holds that $u \in C$ or $v \in C$. Vertex Cover, sometimes called Node Cover, is NP-complete. A straightforward greedy algorithm shows that Vertex Cover is approximable to a ratio 1 (cf. [51]), that is, the greedy algorithm always finds a vertex cover of size at most twice as large as as the optimal one. The simple idea behind the greedy algorithm is to pick any edge from the graph, put both endpoints in the vertex cover, and delete these endpoints together with their incident edges from the graph. Surprisingly, this is very close to the best known bound, which gives a ratio $1 - \log\log|V|/(2\log|V|)$ approximation [6, 17, 47]. However, unless $P = NP$, Vertex Cover has no polynomial time approximation scheme [4] and it is known to be *not* approximable to a ratio 0.1666 [36].

Although Vertex Cover is hard to approximate, it has turned out that it is "easy to parameterize": Vertex Cover has seen quite some history of progress with respect to fixed parameter algorithms (see [24] for details). One of the first results (of mainly theoretical interest) showing its fixed parameter tractability was based on Robertson and Seymour's deep theory of graph minors [28, 29] leading to an $O(n^3)$ algorithm for constant k. Even a linear time algorithm followed, because graphs with "bounded vertex cover" have bounded treewidth [9]. However, more efficient algorithms based on techniques as *bounded search tree* [22] and *reduction to problem kernel* [13] have been obtained. Using maximum matching as a subroutine, Papadimitriou and Yannakakis [53] showed that Vertex Cover admits a polynomial time solution whenever the cover size is $O(\log n)$. Surprisingly, in essence all this already follows from the elementary search tree method described in Mehlhorn's text book on graph algorithms [46, page 216], published before all of the above-mentioned papers. Balasubramanian *et al.* [5] came up with a greatly improved fixed parameter algorithm for Vertex Cover, running in time $O(kn + (1.3248)^k k^2)$. They employ an intricate, improved search tree algorithm. Recently, this result was slightly improved to $O(kn + (1.31951)^k k^2)$ [25]. Note that according to the authors this "tiny difference amounts to a 21% improvement in the running time for $k = 60$." Very recently, the exponential base could be lowered below 1.3, giving an upper bound of $O(kn + (1.2918)^k)$ [50].

In the following subsection, we describe the basic ideas behind the algorithm of Balasubramanian et al. The further improvement was achieved using similar ideas. In particular, studying this concrete problem, the purpose is also to become familiar with the two so far perhaps most successful techniques in designing efficient fixed parameter algorithms—bounded search tree and reduction to problem kernel. Afterwards, in Subsection 5.2, we give one example how to apply this methodology to a problem originating from reconfigurable VLSI—Constraint Bipartite Vertex Cover. This may give sufficient stimulus to pursue further research in this direction.

5.1 General Vertex Cover

Using an intricate, but elementary algorithmic technique, Balasubramanian *et al.* developed a very efficient fixed parameter algorithm to solve Vertex Cover [5]. Let's see how this basically works.

Method 1: reduction to problem kernel.

The general idea of this method, which is fairly generally applicable (not only to vertex cover or graph problems), can be expressed as follows.

1. Reduce the given instance to a new instance whose size is *exclusively* bounded by a function of the parameter k.

2. Perform exhaustive search in the new instance, usually employing an exponential time algorithm.

In this way, we get *Buss' algorithm* for Vertex Cover [13, 23, 24], see Fig. 1. Obviously, the new instance has size bounded by $O(k^2)$.

The correctness of Buss' algorithm relies on the idea that "high-degree-vertices", that is, those of degree $> k$, *must* be part of the vertex cover of size $\leq k$ if one exists. It is not difficult to see, using appropriate subalgorithms, that Buss' algorithm has a running time $O(kn + (2k^2)^k k^2)$. Although in the parameterized world reduction to problem kernel is usually attributed to Buss [23, 24], basically the same technique has been used at least 10 years earlier in VLSI, e.g., by Evans [26]. Reduction to problem kernel is commonly used as some kind of preprocessing to a so-called bounded search tree algorithm, which already can be found in Mehlhorn's textbook [46, page 216].

Input: A graph $G = (V, E)$ and a positive integer k.
Output: A minimum vertex cover of size at most k if one exists.

1. (a) Let H be the set of vertices in V with degree $> k$;
   (b) **if** $|H| > k$ **then** "No size $\leq k$ vertex cover exists"; **exit**;
   (c) Let $G' = (V', E')$ be the graph originating from G by deleting all vertices in H and their incident edges;
   (d) $k' := k - |H|$;
   (e) Delete all isolated vertices in G';
2. **if** $|E'| > kk'$ **then** "No size $\leq k$ vertex cover exists"; **exit**;
3. Perform exhaustive search on G' to find a minimum vertex cover of size k';
4. The minimum vertex cover for G is the minimum vertex cover for G' combined with H.

Figure 1: Buss' algorithm for Vertex Cover—reduction to problem kernel.

Method 2: bounded search tree.

The general idea of this method is to identify a small subset of elements of which at least one must be in *any* feasible solution of the problem. Here comes the application to Vertex Cover based on Mehlhorn's description [46, page 216]. See Fig. 2. It is called *search tree algorithm* and is quite similar in spirit to the greedy approximation algorithm for Vertex Cover, cf. [51]. We use the notion "$G - v$" to express that vertex v and all its incident edges are deleted from G. The time complexity of the algorithm can be easily bounded by $O(2^k n)$.

Using Buss' algorithm as preprocessing phase and directly employing the search tree algorithm, we obtain a vertex cover algorithm running in time $O(kn+2^k k^2)$. However, combining methods 1 and 2 *and* improving on method 2 leads to the result of Balasubramanian *et al.* [5], which we will focus on next.

Input: A graph $G = (V, E)$ and a positive integer k.
Output: A minimum vertex cover of size at most k if one exists.

1. Construct a complete binary tree of height k;
2. Label the root node $(G, \emptyset)$;
3. Recursively label all tree nodes as follows, where (H, S) shall be an already labeled tree node:
   (a) Pick an arbitrary edge (u, v) in graph H;
   (b) Label left child of (H, S) with $(H - u, S \cup \{u\})$;
   (c) Label right child of (H, S) with $(H - v, S \cup \{v\})$;
4. **if** there is a tree node labeled $(\emptyset, S')$ ($\emptyset$ referring to the "empty graph")
   then "S' is a vertex cover of size $\leq k$"
   else "No size $\leq k$ vertex cover exists".

Figure 2: Search tree algorithm for Vertex Cover.

An improved search tree algorithm.

The key idea of Balasubramanian *et al.* [5] to improve the described search tree method with exponential factor 2^k is to do a careful case distinction by distinguishing between the degree of the vertices of the given graph. Observe that factor 2^k actually is the size of the search tree. So, the goal is to decrease the size of the search tree by using a more sophisticated recursion. Before we describe this in more detail, note that by making use of method 1 as a preprocessing phase, w.l.o.g. we can assume that the subsequent search tree algorithm only has to operate on input graphs of size $O(k^2)$. More precisely, what we do is to run the first two steps of the algorithm in Fig. 1 and to replace the second two steps by an improved version of a bounded search tree (Fig. 2).

The basic structure of the improved search tree algorithm is as follows: We distinguish between five cases in the following order, which is given by the degree of the vertices in the graph: First, we deal with “degree-1-vertices”, second, with “degree-2-vertices”, third, with “degree-$\geq$ 5-vertices”, fourth, with “degree-3-vertices”, and, finally, with the remaining graph. Observe that the remaining graph is 4-regular, that is, each vertex has exactly degree 4. To describe all these cases is out of the scope of this paper. To illustrate the fundamental ideas, however, it is sufficient to describe the first two (and most simple) ones.

The degree-1-vertex case is trivial. If a vertex x has only one neighbor y, then to cover the edge between them, it is always advantageous to pick y for the vertex cover set, because if y has more than one neighbor, we cover more edges this way than by choosing x. By always choosing y, a branching of the recursion in the search tree can be avoided, implying a decrease of its size.

The degree-2-vertex case becomes more involved. We distinguish between three subcases. Assume that the considered degree-2-vertex x has neighbors y and z. If there is an edge between y and z, then we avoid any branching of the recursion by always choosing y and z to be included into the vertex cover set. It is not hard to check that this is always optimal in order to cover the edges (x, y), (x, z), and (y, z) and further possibly incident edges to y and z. Subcase 2 addresses the setting where y and z together have at least two neighbors other than x, say a and b. Then with not too much effort (try!) it can be checked that either $\{y, z\}$ or all neighbors of y and z have to be added to the vertex cover set. Thus we get a branching of our recursion. Denoting by $T(k)$ the size of the recursion tree, this branching leads to the recurrence

$$T(k) = 1 + T(k-2) + T(k-3).$$

It is important to emphasize here that this is already one of the worst cases for the improved search tree, that is, the solution of this recurrence already yields the exponential factor $(1.3248)^k$ for the tree size as it is part of the overall result. Finally, subcase 3 (“otherwise”) deals with the situation when y and z together have one neighbor other than x, say a. Then again a branching of the recursion can be avoided by the choice $\{a, x\}$ for the vertex cover set. The optimality of this choice is checked easily.

The complete analysis, involving many more and more complicated cases with, however, the same basic flavor, gives an improved search tree of size $(1.3248)^k$ [5]. All in all, this results in a running time $O(kn + (1.3248)^k k^2)$. Downey *et al.* [25] achieved a reduction of the search tree size from $(1.3248)^k$ to $(1.3195)^k$ by employing a somewhat different reduction to problem kernel, which not only works as preprocessing, but also is applied during the search tree construction. In such a way, they handle graph vertices of degree 1, 2, 3 and of degree greater than k by reduction to problem kernel, so that the search tree only has to take care of vertices degree between 4 and k (the difficult cases being vertices of degree 4 and 5). We refer to Downey *et al.* [25] for any details. The further improvement to search tree size $(1.2918)^k$ [50] again is closer in spirit to Balasubramanian *et al.*, the main innovations occur in the handling of the degree 2, degree 3, and degree 4 vertices. Moreover, an easy observation makes it possible to omit degree 5 vertices from further consideration [50].

What about the potential for improvement of these results? It seems to be quite complicated to do this due to the great number of case distinctions involved. In particular, there is more than one worst case and improving some particular cases may not help in bringing down the overall worst case. However, only elementary combinatorial considerations have been used for this result and maybe with the help of machine support one could still find a better recursion.

5.2 Constraint Bipartite Vertex Cover

Kuo and Fuchs studied the *Constraint Bipartite Vertex Cover (CBVC)* problem, deriving from applications in reconfigurable VLSI [43]. The problem is, given a bipartite graph $G = (V_1, V_2, E)$ and two positive integers k_1 and k_2, are there two subsets $C_1 \subseteq V_1$ and $C_2 \subseteq V_2$ such that $|C_1| \leq k_1$ and $|C_2| \leq k_2$ and each edge from E has at least one endpoint in $C_1 \cup C_2$? In addition, motivated by the applications behind, it is interesting to search for *all* solutions to CBVC with minimal values for the vector $(|C_1|, |C_2|)$. CBVC is *NP*-complete in general [43]. Therefore, in practice, heuristic algorithms are used that not always yield optimal solutions. Since the parameter values k_1 and k_2 can be assumed to be quite small for technological reasons (say $k_1 + k_2$ around 50 all in all), algorithms exponential in k_1 and k_2 may be tolerable as long as the running time is linear in the size of the problem instance.

The affinity between CBVC and Vertex Cover is obvious. However, the existence of *two* parameters in combination with the bipartite nature of the graph means a significant hurdle. So, the Vertex Cover algorithm cannot be translated into this new setting. However, the basic techniques as reduction to problem kernel and bounded search tree again apply. Thus, again based on the degree of vertices, the combination of these two techniques yields an $O((k_1 k_2)n + (1.4656)^{k_1+k_2} k_1 k_2)$ algorithm [30]. Here, the case distinction in comparison with the Vertex Cover case gets less complicated and deals as main cases with "vertices of degree at least three" and "vertices of degree at most two". However, note that the seemingly trivial case of vertices with degree at most two requires some care due to the existence of more than one minimal solution (opposite to the general vertex cover case). In particular, this holds true when generalizing CBVC from 2 to even 3 parameters, yielding so-called "3CBVC," which is motivated by applications from reconfigurable programmable logic arrays [35]. The point here is to partition one of the two vertex sets of the given graph into two subsets. Nevertheless, solutions of efficiency comparable to CBVC can be achieved [30].

It is remarkable that the discussed algorithms for (3)CBVC hide a great potential for heuristic improvements, possibly speeding up the average case running time significantly [30]. A lot of work in this direction, providing a link between theoretical research and applications, still has to be done. It seems to be promising, however, that there are possibilities for mutual exchange: Efficient worst case fixed parameter algorithms may be tuned (using heuristics) with respect to their average case performance and known heuristics may inspire improvements for parameterized algorithms. In addition, besides trying to improve the performance of the proposed (3)CBVC algorithms, there appear to be numerous challenges from VLSI design concerning efficient parameterized algorithms [30], e.g., [34, 35, 40, 42, 43, 44, 57, 61].

6 Maximum Satisfiability problems

Maximum Satisfiability (MaxSat for short) is a problem especially well-known from the field of approximation algorithms [17, 37, 52], having also important practical applications [33]. Hence, many heuristics are in use for MaxSat [7, 11, 33]. The instance is a boolean formula in conjunctive normal form (CNF), the problem is to find a truth assignment that satisfies the most number of clauses. The decision version of MaxSat is NP-complete [31, 51], even if the clauses have at most two literals (so-called Max2Sat). One of the major results in theoretical computer science in recent time shows that if there is a PTAS for MaxSat, then $P = NP$ [4]. On the positive side, it is known that MaxSat is approximable to a ratio 0.3193 [32]. For special cases of MaxSat, better bounds are known: Max$q$Sat is approximable to a ratio $2^{-q}/(1-2^{-q})$ if every clause contains exactly q literals [39], Max3Sat is approximable to a ratio 0.2489 [59], and Max2Sat is approximable to a ratio 0.0741 [27]. On the negative side, Max$q$Sat is not approximable to a ratio $2^{-q}/(1-2^{-q})-\epsilon$ for any $\epsilon > 0$ and $q \geq 3$ and Max2Sat is not approximable to a ratio 0.0476 [36].

The natural parameterized version of MaxSat requires for an algorithm that determines whether at least k clauses of a CNF formula F can be satisfied. Assume that F contains m clauses and n variables. For each F there always exists a truth assignment satisfying at least $\lceil m/2 \rceil$ clauses: simply pick any assignment—either it does or its bitwise complement does. This can be checked in time $O(|F|)$. This observation was used by Cai and Chen [14] to prove that parameterized Max$q$Sat for some constant q is in FPT, implying that every problem in the optimization class $MaxSNP$ is also in FPT. However, their algorithm relies on the boundedness of clauses, which is not necessary for the algorithms described in the following. Of course, one might argue that the proposed parameterization of MaxSat does not make much sense because for $k \leq \lceil m/2 \rceil$ the problem is trivial and for $k > \lceil m/2 \rceil$ one usually cannot speak any longer of a "small parameter value." This is also why Mahajan and Raman [45] introduced a more meaningful parameterization, asking whether at least $\lceil m/2 \rceil + k$ clauses of a CNF formula F can be satisfied. However, the first parameterization still remains of interest, since from a "non-parameterized point of view," an algorithm with running time exponential in m with a small base for the exponential factor can be of interest. So, we firstly stick to this basic parameterization and afterwards very briefly deal with the "more meaningful" parameterization.

Mahajan and Raman [45] presented an algorithm running in time $O(|F| + k^2\phi^k) \approx O(|F| + k^2(1.6181)^k)$ that determines whether at least k clauses of a CNF formula F are satisfiable. This algorithm uses a reduction to problem kernel as well as a bounded search tree.

The *reduction to the problem kernel* relies on the distinction between "large" clauses (i.e., clauses containing at least k literals) and "small" clauses (i.e., clauses containing less than k literals). If F contains at least k large clauses, then it is easy to see that at least k clauses in F can be satisfied. Hence, the subsequent search tree method has only to deal with small clauses. Observe that the size of the remaining "subformula of small clauses" can easily be bounded by $O(k^2)$. This is also owing to the fact that if the number of clauses in F is at least $2k$, then trivially k clauses in F can be satisfied.

Now the *bounded search tree*, here more appropriately called *branching tree*, appears as follows. First, note that we can restrict ourselves to only considering variables that occur both positively and negatively in F, because so-called "pure literals" can always be set true, always increasing the number of satisfied clauses without any disadvantage. The basic technique now is to pick one variable x occurring both positively and negatively in F and then to "branch" into two subformulas $F[x]$ and $F[\bar{x}]$, which arise by setting x to true and false. Clearly, the size of such a branching tree can easily be bounded by 2^k. However, Mahajan and Raman [45] use a further trick, which is basically as follows: Distinguish between two cases. First, if the selected variable occurs exactly twice in F, then one can do a "resolution" avoiding any branching of the recursion:

$$F = (x \vee f_1) \wedge (\bar{x} \vee f_2) \wedge G,$$

where G contains no occurrence of variable x, can be replaced by

$$F' = (f_1 \vee f_2) \wedge G,$$

knowing that one clause in F could be satisfied. Second, if variable x occurs at least three times in F, then we get for the size $T(k)$ of the branching tree the Fibonacci recurrence

$$T(k) \leq 1 + T(k-1) + T(k-2).$$

Altogether, we end up with time complexity $O(|F| + k^2\phi^k)$, where $\phi = (1+\sqrt{5})/2$ is the golden ratio. Independently, in the context of approximation algorithms, basically the same result was also achieved by Dantsin *et al.* [18].

Using many more, carefully designed transformation and splitting rules for propositional formulas, the above result could be improved to time complexity $O(|F| + k^2(1.39995)^k)$ [49]. The fundamental ideas, which all deal with decreasing the size of the branching tree, are as follows. First, there are several "transformation rules", which avoid any branching of the recursion. Besides the described resolution rule, we also have transformation rules called pure literal rule, complementary unit-clause rule, dominating unit-clause rule, small subformula rule, and star rule. Refer to [49] for details. Even more interesting (and more complicated) are the so-called splitting rules, i.e., those rules that lead to a branching of the recursion. The basic idea is to distinguish between the number of occurrences of the variables in formula F. Note that because of some kind of "pre- and postprocessing" done by the transformation rules, we always can restrict attention to variables that occur at least three times in F. Hence, the three main cases now are if a variable x occurs at least 5 times in F, if all variables occur exactly 4 times in F, or if there is some variable that occurs exactly 3 times in F. Obviously, these cover all possibilities. The case of at least 5 variables is quite easy and requires a branching into $F[x]$ and $F[\bar{x}]$. The other two cases, however, are much more complicated and require six, respectively, seven subcases. For the purpose of illustration, we just give one of them, namely, the case that variable x occurs exactly three times in F and F is as follows:

$$F = (\bar{x} \vee \bar{l} \vee \dots) \wedge (x \vee l \vee \dots) \wedge (x \vee \dots) \wedge (l \vee \dots) \wedge \dots$$

That is, there is another literal l occurring together with $\bar{x}$ as $\bar{l}$ in one clause, occurring together with x as l in a second clause, and occurring in at least one clause not containing variable x. Then the rule "T2" [49] says that we branch (or split) into $F[l]$ and $F[\bar{l}]$. It is not hard to see that in the step from F to $F[l]$ at least two clauses and a further one by applying the resolution rule afterwards are satisfied, and in the step from F to $F[\bar{l}]$ at least one clause and two further clauses by applying the pure literal rule applied to x are satisfied.

All in all, it can be shown that all cases lead to a recursion that yields a branching tree size that can be bounded by $(1.3995)^k$. Interestingly, there is a slightly "better" result if we measure the complexity not in the parameter k of number of clauses to be satisfied, but in the total number m of clauses in F. Then the branching tree size can be bounded by $(1.3972)^m$. Again we refer to [49] for any further details. It is worth noting that in particular the development of the MaxSat algorithm shows how seemingly tight the relation between fixed parameter tractability and the in some sense more general topic of worst case upper bounds for NP-hard problems is. So, both for Vertex Cover and for MaxSat, the best known worst case algorithms also yield the best known parameterized algorithms.

We only mention in passing that the parameterization of MaxSat requiring the satisfiability of at least $\lceil m/2 \rceil + k$ clauses is led back to the case considered above by Mahajan and Raman, thus obtaining a time complexity of $O(|F| + k^2\phi^{6k}) \approx O(|F| + k^2(17.9443)^k)$. Plugging in the above described improvements [49], we immediately get $O(|F| + k^2(1.3995)^{6k}) \approx O(|F| + k^2(7.5135)^k)$. Furthermore, Mahajan and Raman also study the MaxCut problem [45]. Here, analogous parameterizations as for MaxSat exist. The "conventional" parameterization requiring a cut of size at least k can be led back to a Max2Sat problem [45, 60] and thus can also be improved basically using the above described results [49]. Note that for the Max2Sat problem, the sketched algorithm implies a time bound of $O((1.2722)^m)$: The Maximum Cut problem (MaxCut) for undirected graphs is as follows: Given a graph $G = (V, E)$ on n vertices and m edges and an integer k, is there a set $S \subset V$ such that at least k edges of G have one endpoint in V and one endpoint in $V - S$? Mahajan and Raman prove that MaxCut can be solved in time $O(m + n + k4^k)$ [45]. Using the reduction to Max2Sat (also see "Method 2" in [45]), this improves to a time bound of $O(m+n+k^2 1.2722^{4k}) = O(m+n+k^2 2.6196^k)$, thus basically decreasing the exponential base from 4 to 2.6196.

7 Conclusion

The purpose of this paper was not to give a complete survey on how to develop algorithms that prove fixed parameter tractability, but, by taking a restricted viewpoint, we concentrated on the techniques of reduction to problem kernel and, even more importantly, bounded search trees. We freely omitted referencing to Robertson-Seymour theorems or bounded treewidth machinery, other helpful tools in proving fixed parameter tractability. For an extensive overview of parameterized complexity covering these aspects and, moreover, the deep structural theory behind parameterized intractability, we refer to the new monograph by Downey and Fellows [24]. The main intention of this paper was to give an incomplete, but easily understandable review on interesting algorithmic aspects of the field and, in particular, to point out the potential for future algorithmic research topics in this direction. The ultimate goal of this work could be termed as putting the reader into the position to pursue research on efficient parameterized algorithms, maybe even without the need of studying the extensive literature on parameterized complexity.

Also, we concentrated on two core problems: Vertex Cover and MaxSat. Besides the open problems in direct relation with these two, we strongly believe that there is a big potential for research on efficient parameterized algorithms in fields like VLSI design, computational biology, logic, data bases, and several others. Practical algorithms from these fields often use heuristic ideas, and may yield new insight for parameterized algorithm design. It is also promising to test parameterized algorithms in practical applications, e.g., combining or enriching them with some heuristics. Research on efficient parameterized algorithms thus could mean entering a field that offers a link between computer science theory and practice.

Acknowledgment. I'm grateful to Henning Fernau for many pointers to the literature. I appreciate having been invited by Jan Kratochvíl and Jarik Nešetřil to give lectures on parameterized complexity at DIMATIA, Prague. In addition, I'm also grateful to Jiří Wiedermann for inviting me to a talk on the presented topic, which together with the lecture notes served as a basis for this paper. Thanks go also to Rod Downey for providing me with a preliminary version of the new monograph [24]. Finally, I'm grateful to Henning Fernau and Ton Kloks for helpful comments on previous drafts of the paper.

References

[1] K. A. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73:235–276, 1995.

[2] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33d IEEE Conference on Foundations of Computer Science*, pages 14–23, 1992.

[5] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[6] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Disc. Math.*, 25:27–46, 1985.

[7] R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2:Article 2, 1997.

[8] A. Blummer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36:929–965, 1989.

[9] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[10] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A

of *Handbook of Theoretical Computer Science*, chapter 14, pages 757–804. Elsevier, 1990.

[11] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Mar. 1997. To appear in *Journal of Combinatorial Optimization.*

[12] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1994.

[13] J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

[14] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54:465–474, 1997.

[15] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetical progression. *J. Symbolic Computations*, 9:251–280, 1990.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[17] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, Apr. 1997.

[18] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. Approximation algorithms for Max SAT: a better performance ratio at the cost of a longer running time. Technical Report PDMI preprint 14/1998, Steklov Institute of Mathematics at St. Petersburg, 1998.

[19] R. G. Downey, P. Evans, and M. R. Fellows. Parameterized learning complexity. In *6th Annual Conference on Learning Theory, COLT'93*, pages 51–57. ACM Press, 1993.

[20] R. G. Downey and M. R. Fellows. Fixed parameter tractability and completeness III. In *Complexity Theory: Current Research, Edited by K. Ambos-Spies, S. Homer, and U. Schöning, Cambridge University Press*, pages 191–225. 1993.

[21] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, August 1995.

[22] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141:109–131, 1995.

[23] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[24] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1998.

[25] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In F. Roberts, J. Kratochvíl, and J. Nešetřil, editors, *The Future of Discrete Mathematics: Proceedings of the First DIMATIA Symposium, June 1997*, AMS-DIMACS Proceedings Series. AMS, 1998. To appear. Available through http://www.inf.ethz.ch/personal/stege.

[26] R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int. Test Conf.*, pages 49–55, 1981.

[27] U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *3d IEEE Israel Symposium on the Theory of Computing and Systems*, pages 182–189, 1995.

[28] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1987.

[29] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *J. ACM*, 35:727–739, 1988.

[30] H. Fernau and R. Niedermeier. Efficient algorithms for constraint bipartite vertex cover problems. Manuscript, July 1998.

[31] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.

[32] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995.

[33] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[34] N. Hasan and C. L. Liu. Minimum fault coverage in reconfigurable arrays. In *18th Int. Symp. on Fault-Tolerant Computing Systems (FTCS'88)*, pages 348–353. IEEE Computer Society Press, 1988.

[35] N. Hasan and C. L. Liu. Fault covers in reconfigurable PLAs. In *20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS'90)*, pages 166–173. IEEE Computer Society Press, 1990.

[36] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[37] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997.

[38] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22:463–468, 1975.

[39] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9:256–278, 1974.

[40] J. H. Kim and S. M. Reddy. On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement. *IEEE Transactions on Computers*, 38(4):515–525, Apr. 1989.

[41] P. G. Kolaitis and M. N. Thakur. Approximation properties of NP minimization classes. *J. Comput. Syst. Sci.*, 50:391–411, 1995.

[42] S.-Y. Kuo and I.-Y. Chen. Efficient reconfiguration algorithms for degradable VLSI/WSI arrays. *IEEE Transactions on Computer-Aided Design*, 11(10):1289–1300, 1992.

[43] S.-Y. Kuo and W. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, 4:24–31, Feb. 1987.

[44] C. P. Low and H. W. Leong. A new class of efficient algorithms for reconfiguration of memory arrays. *IEEE Transactions on Computers*, 45(5):614–618, 1996.

[45] M. Mahajan and V. Raman. Parametrizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, ECCC Trier, 1997. To appear in *Journal of Algorithms*.

[46] K. Mehlhorn. *Graph algorithms and NP-completeness*. Heidelberg: Springer, 1984.

[47] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.

[48] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[49] R. Niedermeier and P. Rossmanith. New upper bounds for MaxSat. Technical Report KAM-DIMATIA Series 98-401, Faculty of Mathematics and Physics, Charles University, Prague, July 1998. Submitted for publication.

[50] R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover—further improved. Manuscript, submitted for publication, September 1998.

[51] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[52] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43:425–440, 1991.

[53] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *J. Comput. Syst. Sci.*, 53:161–170, 1996.

[54] V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I–1–I–18, June 1997.

[55] N. Robertson and P. D. Seymour. Graph minors—a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge University Press, 1985.

[56] S. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23:116–127, 1976.

[57] M. D. Smith and P. Mazumder. Generation of minimal vertex covers for row/column allocation in self-repairable arrays. *IEEE Transactions on Computers*, 45(1):109–115, 1996.

[58] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.

[59] L. Trevisan, G. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th IEEE Conference on Foundations of Computer Science*, pages 617–626, 1996.

[60] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17:475–502, 1994.

[61] L. Youngs and S. Paramanandam. Mapping and repairing embedded-memory defects. *IEEE Design and Test*, 14(1):18–24, 1997.